UNIVERSIDAD DE ALCALÁ
Ingeniería en Sistemas de Información

Estructuras de Datos
Curso 2020-2021

PECL2
(SIMULACIÓN DEL CONTROL DE ACCESO A UN EVENTO DEPORTIVO)

ADINA MURG -

VICTORIA LORENA ORDENES ORBEGOZO -

ÍNDICE DE CONTENIDO:

1. Detalles y justificación de la implementación

En esta sección comentaremos las operaciones y especificación del ABB que se ha empleado a lo largo de esta práctica, además de las nuevas operaciones de la clase Gestor.

Se obviarán los métodos propios de Pila, Cola y Lista ya que vienen descritos en la PECL1, por lo que nos enfocaremos principalmente en los árboles, además de que no han sufrido modificaciones (salvo Lista, que se ha mejorado un método y la cuál describiremos más adelante).

1.1 Arbol

espec ARBOL_BUSQUEDA_BINARIO

usa NATURAL, BOOL

generos abb

operaciones

insertar: aficionado -> abb

altura: abb -> natural

buscar: abb -> aficionado

aficionadosPares: abb -> natural

mostrarHojas: abb -> aficionado

existe: natural -> bool

mostrar: String -> abb

inOrden: abb -> aficionado

contarAficionados: abb -> natural

nodosHojas: abb -> aficionado

comprobarExistencia: natural, abb ->bool

eliminarNodo: natural -> abb

maximo: Aficionado, abb -> aficionado

minimo: Aficionado, abb -> aficionado

vacio: abb -> bool

eliminar: natural -> abb

maximoId: abb-> Aficionado

minimoId: abb-> Aficionado

1.2 Nodo Arbol

espec NODOARBOL

usa NATURAL, BOOL

generos anodo

1.3 Gestor

espec GESTOR

usa PILA, COLA, LISTA, AFICIONADO, ARBOL

generos gestor

operaciones

(todas las anteriores de PECL1 además de las siguientes:)

dibujarAbb: abb -> abb

sociosOrdenados: abb -> aficionado

simpatizantesOrdenados: abb -> aficionado

crearArbol: -> abb

todosAficionados: abb -> aficionado

mostrarAficionadosPares: abb -> aficionado

buscar: abb -> aficionado

mostrarHojas: abb -> aficionado

eliminarNodo: natural -> abb

var

colaSocios, colaSimpatizantes: cola

listaSocios: lista

pilaAficionados: pila

arbolAficionados: arbol

2. Solución adoptada: Descripción de Dificultades Encontradas

SOLUCIÓN ADOPTADA:

- Se realizó una mejora de la PECL1, corrigiendo el método de encontrar el mayor y menor de las listas, haciéndolo más correcto y eficiente.
- A medida que realizamos la práctica, fuimos mejorando y adaptando métodos para que fueran más correctos, como ejemplo, una comparación del método de mostrar aficionados, socios y simpatizantes por separado frente a un nuevo método que engloba los 3 anteriores.

Anterior:

```
 99  void Arbol::mostrarSimpatizantes(){
100          anodo aux;
101          aux=raiz;
102          aux=aux->der;
103          InOrden(aux);
104  }
105
106  void Arbol::mostrarAficionados(){
107          anodo aux;
108          aux=raiz;
109          InOrden(aux);
110  }
```

Actual:

```
46  void Arbol::mostrar(string opcion){
47      anodo aux;
48      aux=raiz;
49      if(opcion=="socio"){
50          aux=aux->izq;
51          //InOrden(aux);
52      }
53      if(opcion=="simpatizante"){
54          aux=aux->der;
55          //InOrden(aux);
56      }
57      if(opcion=="aficionado"){}
58          //InOrden(aux);}
59      InOrden(aux);
60  }
```

- Dado que hacemos uso de la recursividad en esta práctica, hemos optado por usar 2 métodos para cada tarea a realizar, como por ejemplo:

```
174 void Arbol::nodosHojas(anodo aux)
175 {
176     if(!aux) { // Si el nodo es Null
177         return;
178     }
179     if((aux->izq == NULL) && (aux->der == NULL)) { // Si no hay elementos en la izquierda y la
    derecha
180         aux->a1.mostrar();
181         return;
182     }
183     if(aux->der) {
184         nodosHojas(aux->der);
185     }
186     if(aux->izq) {
187         nodosHojas(aux->izq);
188     }
189 }
190
191 void Arbol::mostrarHojas()
192 {
193     anodo aux;
194     aux = raiz;
195     nodosHojas(aux);
196 }
```

El método mostrarHojas (el cuál se usará en Gestor) llamará al método recursivo nodosHojas (operará en Árbol).

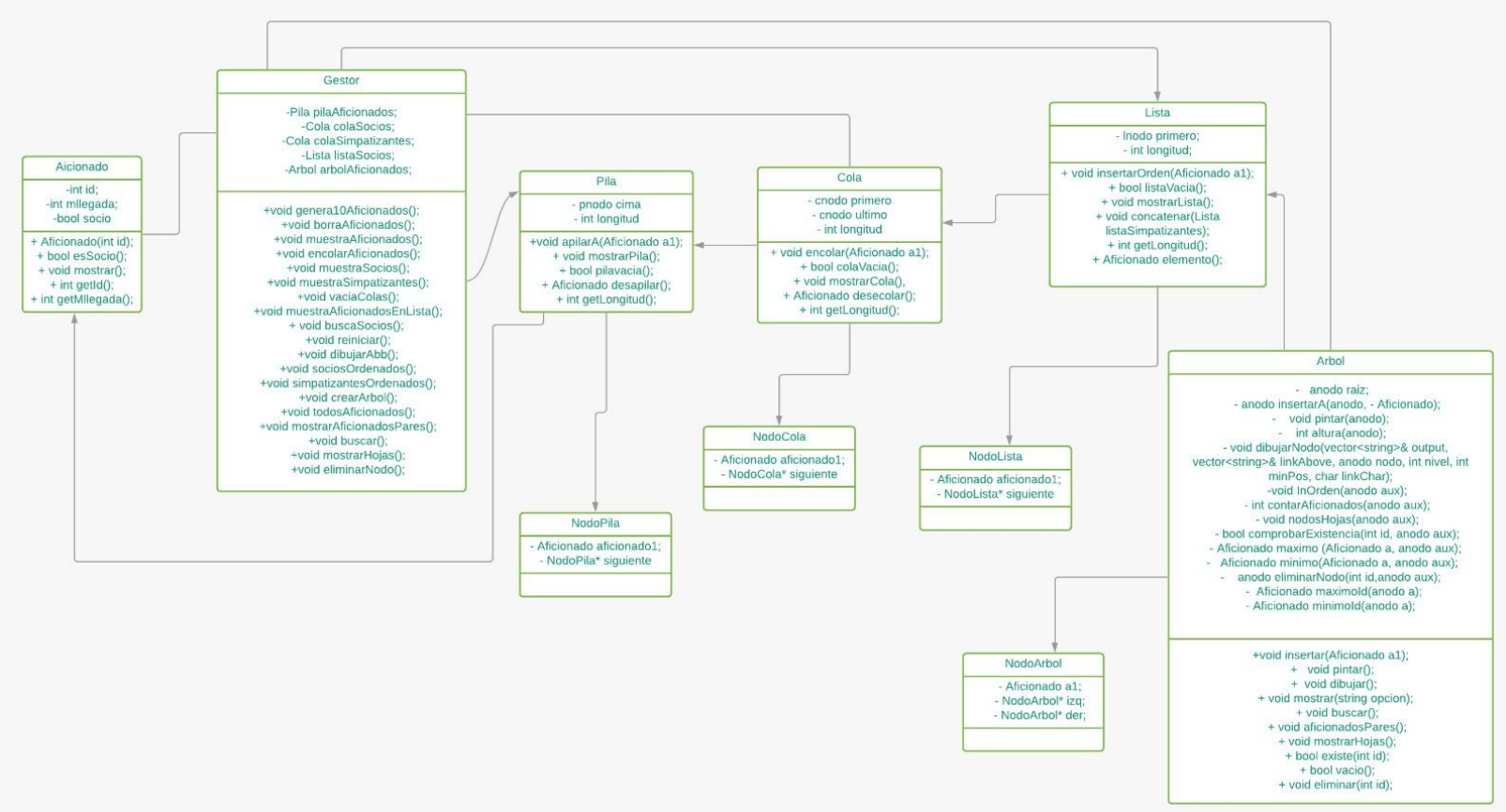
DIFICULTADES ENCONTRADAS:

La principal dificultad encontrada es la de borrar un nodo del árbol. Lo primero que decidimos, es comprobar que el elemento (el id) a borrar existe en el árbol. Una vez hayamos comprobado su existencia, procederemos al borrado. Pese a haber probado las 2 variantes del código de teoría (habiéndose adaptado a nuestro código), nos encontramos con problemas como:

- el borrado del elemento no se hace de forma correcta, el id se borra pero se sustituye por datos basura
- algunos valores no se actualizan correctamente en el árbol (no se actualiza la raíz con el valor correspondiente)

Finalmente logramos resolver el problema, con ayuda del ejercicio de teoría y aplicando de nuevo otra lógica que resultó ser la correcta.

3. Diagrama UML



*Para mayor resolución, consultar el PDF con el diagrama adjunto.

4. Explicación de los métodos más destacados

Estos son los métodos que consideramos más destacados, ya que son de los más relevantes y que más dificultad nos ha costado.

Clase	Nombre	Argumentos	Retorno	Descripción
Arbol	buscar	--	--	Este método nos permite mostrar a los aficionados (socios y simpatizantes) que han llegado más pronto y más tarde al estadio. Dentro de este método llamamos a la función máximo y mínimo.
Arbol	máximo	Nodo Arbol	Aficionado	Este método buscará en la rama correspondiente, izquierda para los socios y derecha para los simpatizantes. Al buscar por el campo minuto de llegada hay que recordar que tenemos un ABB de búsqueda por el campo id. Por lo tanto tenemos que consultar todas las ramas del árbol para quedarnos con el aficionado (socio o simpatizante) que más tarde ha llegado.
Arbol	mínimo	Nodo Arbol	Aficionado	Este método seguirá la misma lógica que la función máxima que hemos explicado anteriormente, solo que en este caso recorreremos todas las ramas y nos quedaremos con el aficionado (socio o simpatizante) que más pronto ha llegado.
Arbol	aficionadosPares	--	--	Este método llamará a otra función llamada contarAficionados. De acuerdo a la lógica de nuestro programa, los aficionados pares se encuentran en la rama izquierda de nuestro ABB.
Arbol	contarAficionados	Nodo Arbol	int	Este método recursivo recibirá por parámetro un nodo árbol apuntando a la rama izquierda del árbol

				que iremos recorriendo, sumando 1 hasta llegar un nodo vacío para saber el número de elementos totales de esa rama izquierda.
Arbol	mostrarHojas	--	--	Este método creará un puntero hacia la raíz del árbol, que luego se lo pasaremos por parámetro a una función llamada nodosHojas.
Arbol	nodosHojas	Nodo Arbol	--	Este método recibe un puntero auxiliar que nos permitirá recorrer el árbol para llegar a las nodos hojas y mostrarlos por pantalla.
Arbol	existe	int	bool	Este método recibe un id. Dentro de esta función junto con un nodo árbol apuntando a la raíz del ABB se lo pasaremos por parámetros a otra función llamada comprobarExistencia.
Arbol	comprobarExistencia	int	bool	Este método nos permite ir recorriendo el árbol de forma recursiva gracias al id que queremos verificar y el puntero auxiliar. Como es un ABB ordenado por el campo id, podemos recorrerlo de la forma más eficiente posible para saber si el id está o no en el árbol.
Arbol	mostrar	string	--	Este método tiene un submenú que nos permitirá elegir entre socios, simpatizantes o aficionados según el string que reciba. Desde esta función llamaremos al método recursivo inorden.

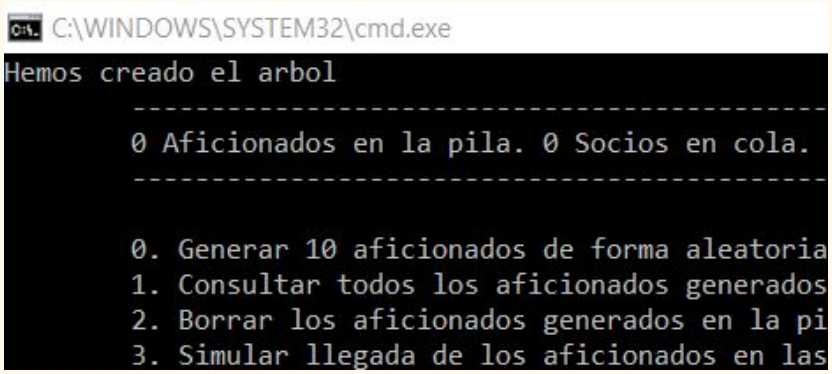
Arbol	InOrden	Nodo Arbol	--	Esta función recibirá un nodo árbol que le permitirá recorrer el ABB que sea en inorden es decir desde izquierda, raíz y derecha.
Arbol	eliminar	int	--	Esta función recibe el id del Aficionado que queremos eliminar y que llamará a otra función eliminarNodo.
Arbol	eliminarNodo	int, Nodo Arbol	Nodo Arbol	Esta función recursiva eliminará el nodo donde se encuentra el aficionado que queremos eliminar.

5. Explicación del comportamiento del programa

Las primeras opciones del programa están contempladas en el manual de la práctica anterior. Empezamos a partir de la opción 7 cuando los elementos se encuentran en la lista ordenados por minuto de llegada. Hemos realizado la simulación con 10 aficionados.

Opción X:

En esta opción mostraremos un mensaje cuando se crea el árbol.

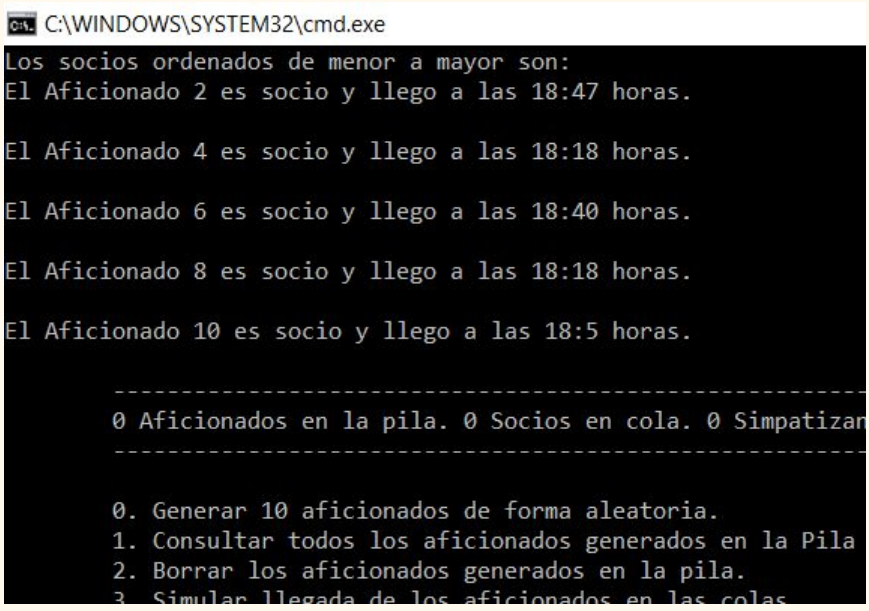


Opción A:

En esta opción mostramos el árbol que se ha creado y en caso de pulsar esta opción con el árbol vacío (solo con el nodo ficticio) aparecería un mensaje de alerta diciendo que el árbol está vacío.

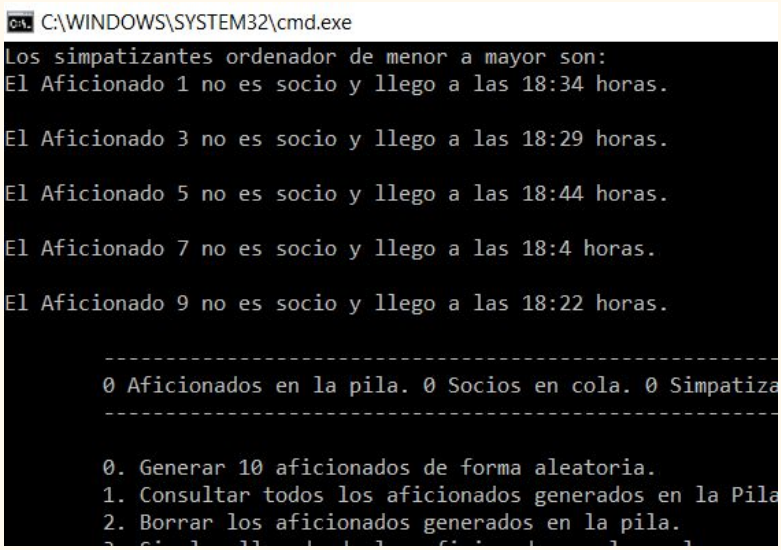
Opción B:

Mostrarán los socios ordenados de menor a mayor por el campo **ID.** Aquí también tenemos contemplado el caso en el cual el árbol esté vacío, y por lo tanto esta opción no se puede ejecutar ya que aparece un mensaje indicandolo.



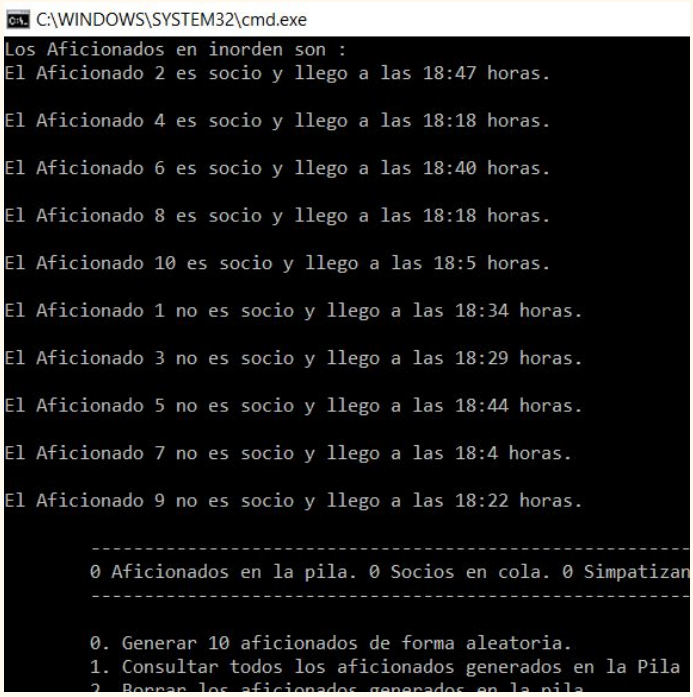
Opción C:

Mostrará los simpatizantes de menor a mayor por el campo **ID.** Aquí también tenemos contemplado el caso de que el árbol esté vacío, y por lo tanto esta opción no se puede ejecutar ya que aparece un mensaje indicandolo.



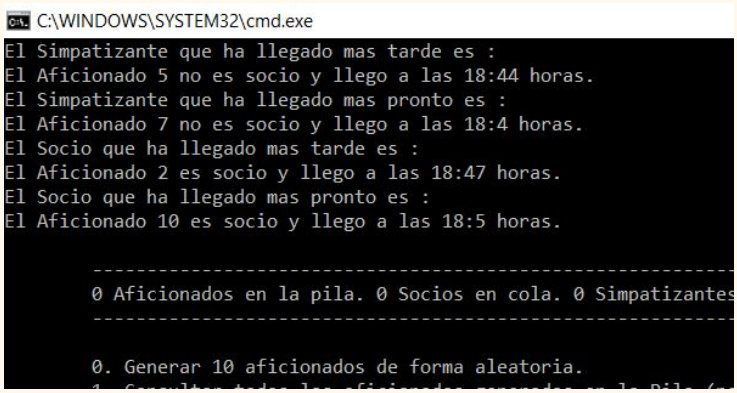
Opción D:

Mostrará a todos los aficionados (socios y simpatizantes) de menor a mayor por el campo **ID.** Aquí también tenemos contemplado el caso de que el árbol esté vacío, y por lo tanto esta opción no se puede ejecutar ya que aparece un mensaje indicandolo.



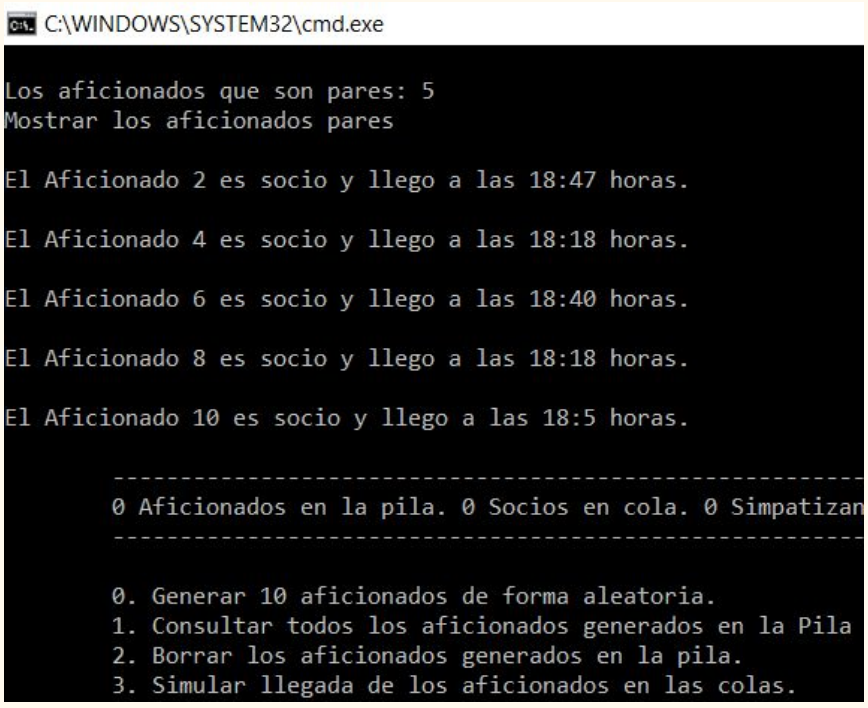
Opción E:

Mostrará los socios y aficionados que más pronto y tarde han llegado al estadio.



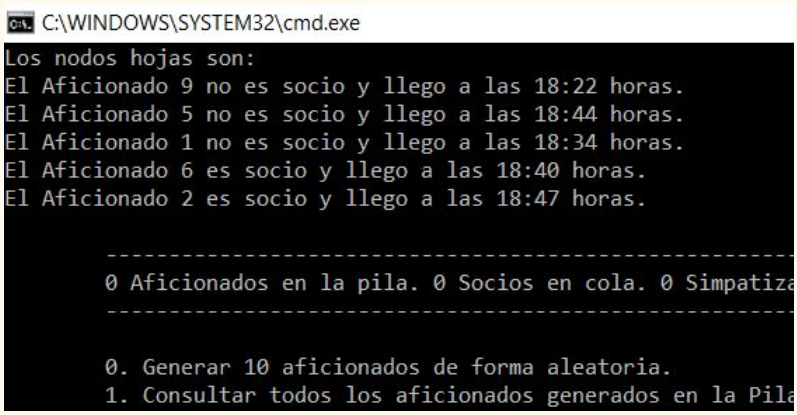
Opción F:

Mostrará el número de aficionados pares que hay en nuestro árbol y también aparecerá toda la información de ellos.



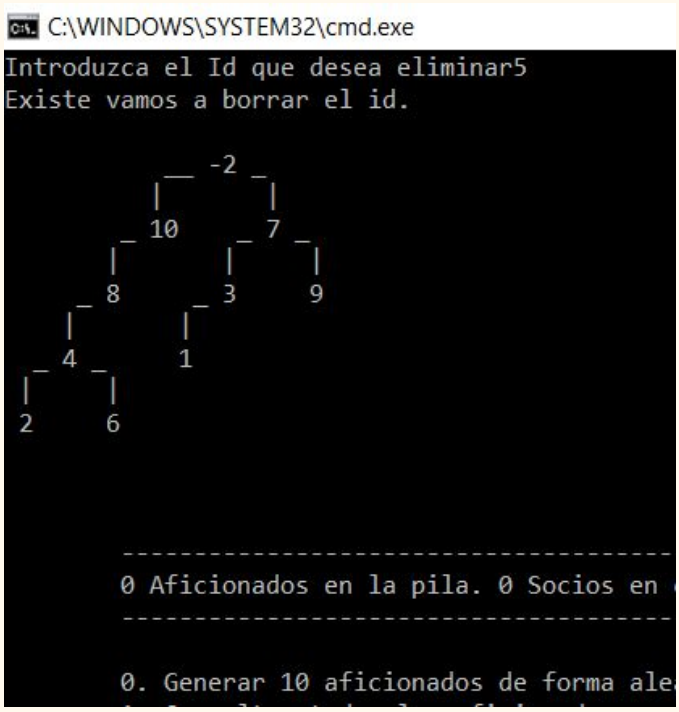
Opción G:

Mostrará la información de los aficionados que sean hojas en nuestro ABB.



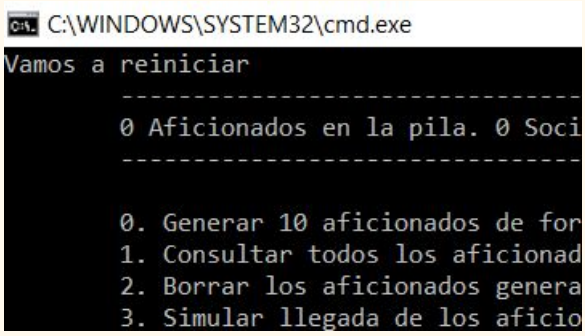
Opción H:

Eliminaremos el nodo del ABB.



Opción 9:

En esta opción reiniciamos todos las estructuras de datos que utilizamos en el sistema.



6. Bibliografía

Apuntes, diapositivas, ejercicios y tutorías de clase:

BlackBoard - Estructuras de Datos.

Herramienta para crear el diagrama UML:

https://app.lucidchart.com/users/login#/login